Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ №1**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. В. Сидоренко

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

**Цель работы**

Написать программу по заданному ниже ТЗ, используя любой язык программирования.

**Описание задачи**

**Поиск патентов**

В системе есть n агентов. У каждого агента есть целевой набор патентов (может быть представлен как строка, глобальный идентификатор и тд.) Аt, который ему нужно собрать для реализации некоторой глобальной задачи. Его целью является сбор всех элементов своего целевого набора.

В начале моделирования каждому агенту выдается непустое множество патентов Аs. Они могут быть как из целевого набора, так и не из него.

В процессе моделирования агенты общаются между собой. Целью коммуникации является обмен патентами. При коммуникации с другим агентом, текущий спрашивает есть ли у другого один из нужных ему патентов. Если нет, то раунд коммуникации завершается. Если у другого агента есть нужный патент, то он готов им поделиться только в обмен на один из нужных для себя. Если у текущего агента есть один из подходящих для второго патентов, обмен завершается. Если агент собрал свой целевой набор, то он может безвозмездно делиться своими патентами.

Цикл моделирования заканчивается, когда у всех агентов собран целевой набор. В результате необходимо вывести список агентов с размером их целевого набора и количеством итераций и раундов коммуникаций, которые ему потребовались для его сбора.

**Ремарки**

Раундом коммуникации считается общение агента с агентом. Например, агент 1 пообщался с агентом 2 и агентом 3. Тогда у агента 1 будет два раунда, а у второго и третьего по одному.

Для генерации начальных условий агента (целевой набор и множество патентов) можно сначала сгенерировать каждому агенту свой набор, затем объединить получившиеся множества в одно, и раздать его в случайном порядке. По итогам раздачи размеры множеств А, каждого агента должны быть примерно одинаковыми.

**Описание решения**

*1. Инициализация системы агентов:*

В начале симуляции создается фиксированное количество агентов, каждый из которых имеет:

* Целевой набор патентов (представлен в виде списка символов), который агент должен собрать.
* Начальное множество патентов (часть патентов может быть полезной для агента, часть — нет).
* Число итераций и число раундов коммуникации — эти параметры отслеживают, сколько итераций прошло и сколько раз агент участвовал в обменах.

Класс Agent:

* Каждый агент создается с уникальным ID и набором патентов. Целевой набор генерируется случайно из строки символов "abcdefghijklmnopqrstuvwxyz", состоящей из 26 символов.
* Агент также хранит два списка: важные патенты (те, которые ему нужны) и ненужные патенты (те, которые он готов обменять).

Методы агента:

* getInfo(): Выводит текущее состояние агента — его целевой набор, собранные патенты и ненужные патенты.
* obtainingAPatent(): Агент получает патент. Если он полезен, то добавляется в список важных патентов, если нет — в список ненужных.
* checkingTask(): Возвращает информацию о том, какие важные патенты еще не собраны.
* swap(): Обновляет состояние агента при обмене патентами.
* winInfo(): По завершении симуляции выводит информацию о том, сколько итераций и обменов потребовалось агенту для сбора полного набора патентов.

*2. Создание начальных условий:*

Функция createAgent() отвечает за:

* Создание агентов с их целевыми наборами.
* Генерацию общего множества патентов для всех агентов (из их целевых списков).
* Случайное распределение патентов между агентами так, чтобы у каждого было примерно одинаковое количество патентов (как нужных, так и ненужных).
* После инициализации система готова к проведению симуляции обменов патентами.

*3. Коммуникация между агентами:*

Ключевая часть решения — это обмен патентами между агентами в ходе симуляции. Здесь используется объект MessageBus, который управляет коммуникацией и обменом между агентами.

Процесс коммуникации:

Итерации:

* Каждый цикл симуляции называется итерацией. В каждой итерации агенты обмениваются патентами с целью собрать свой целевой набор.
* В начале каждой итерации вызывается метод consilium() для организации обмена между агентами.

Обмен патентами:

* Агент запрашивает у другого агента нужный ему патент. Если патент есть, второй агент ищет патент, который нужен ему, у первого агента.
* Если такой патент находится, происходит обмен.
* Если агент уже собрал весь свой целевой набор, он может безвозмездно делиться патентами с другими.

Проверка завершения задачи:

* Каждый агент постоянно проверяет, собрал ли он полный набор патентов. Если это так, он выходит из активной фазы обмена.
* Симуляция продолжается до тех пор, пока все агенты не соберут свои целевые наборы.

Раунды коммуникаций:

* Каждый раунд коммуникации фиксируется: сколько раз агент пообщался с другим агентом. Например, если агент 1 пообщался с агентами 2 и 3, у него будет два раунда, а у второго и третьего — по одному.

*4. Цепочка обменов:*

Когда прямой обмен между двумя агентами невозможен, система пытается найти цепочку обменов — последовательность агентов, через которых можно передать патенты для достижения цели. Эта часть особенно важна, так как позволяет избежать тупиков, где агенты не могут напрямую обменяться патентами.

Алгоритм поиска цепочки:

* Используется метод findChain(), который пытается найти такую последовательность агентов, при которой патенты могут передаваться по цепочке, начиная с одного агента и заканчивая другим.
* Если цепочка находится, происходит обмен патентами по всей цепочке, и агенты обновляют свое состояние.

*5. Завершение симуляции:*

Когда все агенты собрали свои целевые наборы, симуляция завершена. В конце выводится информация о каждом агенте:

* Его целевой набор.
* Количество итераций и раундов коммуникаций, которые потребовались для завершения задачи.

*Основные моменты реализации:*

* Обмен патентами: Каждый агент участвует в коммуникации с целью получить нужный патент, при этом всегда есть возможность найти цепочку обменов, если прямой обмен невозможен.
* Отслеживание состояния агентов: За каждым агентом ведется учет количества итераций и обменов.
* Поиск цепочек: Для более сложных случаев предусмотрен алгоритм поиска цепочки агентов, которые могут обменяться патентами по кругу.

**Код программы**

Agent.java

import java.util.\*;  
import java.util.stream.Collectors;  
  
public class Agent {  
  
 private static final String *TARGET\_STRING* = "abcdefghijklmnopqrstuvwxyz";  
  
 private static final int *LENGTH\_TARGET\_TASK* = 5;  
  
 private int id;  
 private List<String> targetTask;  
 private List<String> importantPatents;  
 private List<String> uselessPatents;  
 private int numberOfIteration;  
 private int numberOfCommunications;  
  
 public Agent(int id) {  
 this.id = id;  
 Random random = new Random();  
  
 this.targetTask = random.ints(*LENGTH\_TARGET\_TASK*, 0, *TARGET\_STRING*.length())  
 .mapToObj(i -> String.*valueOf*(*TARGET\_STRING*.charAt(i)))  
 .sorted()  
 .collect(Collectors.*toList*());  
  
 this.importantPatents = new ArrayList<>();  
 this.uselessPatents = new ArrayList<>();  
 this.numberOfIteration = 0;  
 this.numberOfCommunications = 0;  
 }  
  
 public void getInfo() {  
 Map<String, Long> missingImportantPatents = checkingTask();  
  
 List<String> misImpPatList = missingImportantPatents.entrySet().stream()  
 .flatMap(entry -> Collections.*nCopies*(entry.getValue().intValue(), entry.getKey()).stream())  
 .sorted()  
 .collect(Collectors.*toList*());  
  
 System.*out*.println("Агент: id = " + id + "; ЦЕЛЕВОЙ НАБОР ПАТЕНТОВ: " + targetTask  
 + "; УЖЕ СОБРАЛ: " + importantPatents  
 + "; ЕЩЁ НУЖНЫ: " + uselessPatents  
 + ", ГОТОВ ОТДАТЬ: " + misImpPatList);  
 }  
  
  
 public void obtainingAPatent(String patent) {  
  
 long countPatent = targetTask.stream().filter(p -> p.equals(patent)).count();  
  
 if (countPatent > 0 && importantPatents.stream().filter(p -> p.equals(patent)).count() < countPatent) {  
 importantPatents.add(patent);  
 Collections.*sort*(importantPatents);  
 } else {  
 uselessPatents.add(patent);  
 Collections.*sort*(uselessPatents);  
 }  
  
 }  
  
  
 public void swap(String patent) {  
 this.numberOfCommunications++;  
 this.obtainingAPatent(patent);  
 }  
  
 public void updateState() {  
 this.numberOfIteration++;  
 }  
  
  
 public Map<String, Long> checkingTask() {  
  
 Map<String, Long> targetPatentCount = targetTask.stream()  
 .collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*()));  
  
 Map<String, Long> collectedPatentCount = importantPatents.stream()  
 .collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*()));  
  
 Map<String, Long> missingPatents = new HashMap<>();  
  
 for (Map.Entry<String, Long> entry : targetPatentCount.entrySet()) {  
  
 long collectedCount = collectedPatentCount.getOrDefault(entry.getKey(), 0L);  
  
 if (collectedCount < entry.getValue()) {  
 missingPatents.put(entry.getKey(), entry.getValue() - collectedCount);  
 }  
 }  
 return missingPatents;  
 }  
  
  
 public boolean checkCompletion() {  
 return checkingTask().isEmpty();  
 }  
  
 public void winInfo() {  
 System.*out*.println("Агенту " + id + " == " + targetTask  
 + " потребовалось " + numberOfIteration + " ИТЕРАЦИЙ и "  
 + numberOfCommunications + " ОБМЕНОВ");  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public List<String> getTargetTask() {  
 return targetTask;  
 }  
  
 public List<String> getImportantPatents() {  
 return importantPatents;  
 }  
  
 public List<String> getUselessPatents() {  
 return uselessPatents;  
 }  
  
 public int getNumberOfIteration() {  
 return numberOfIteration;  
 }  
  
 public int getNumberOfCommunications() {  
 return numberOfCommunications;  
 }  
}

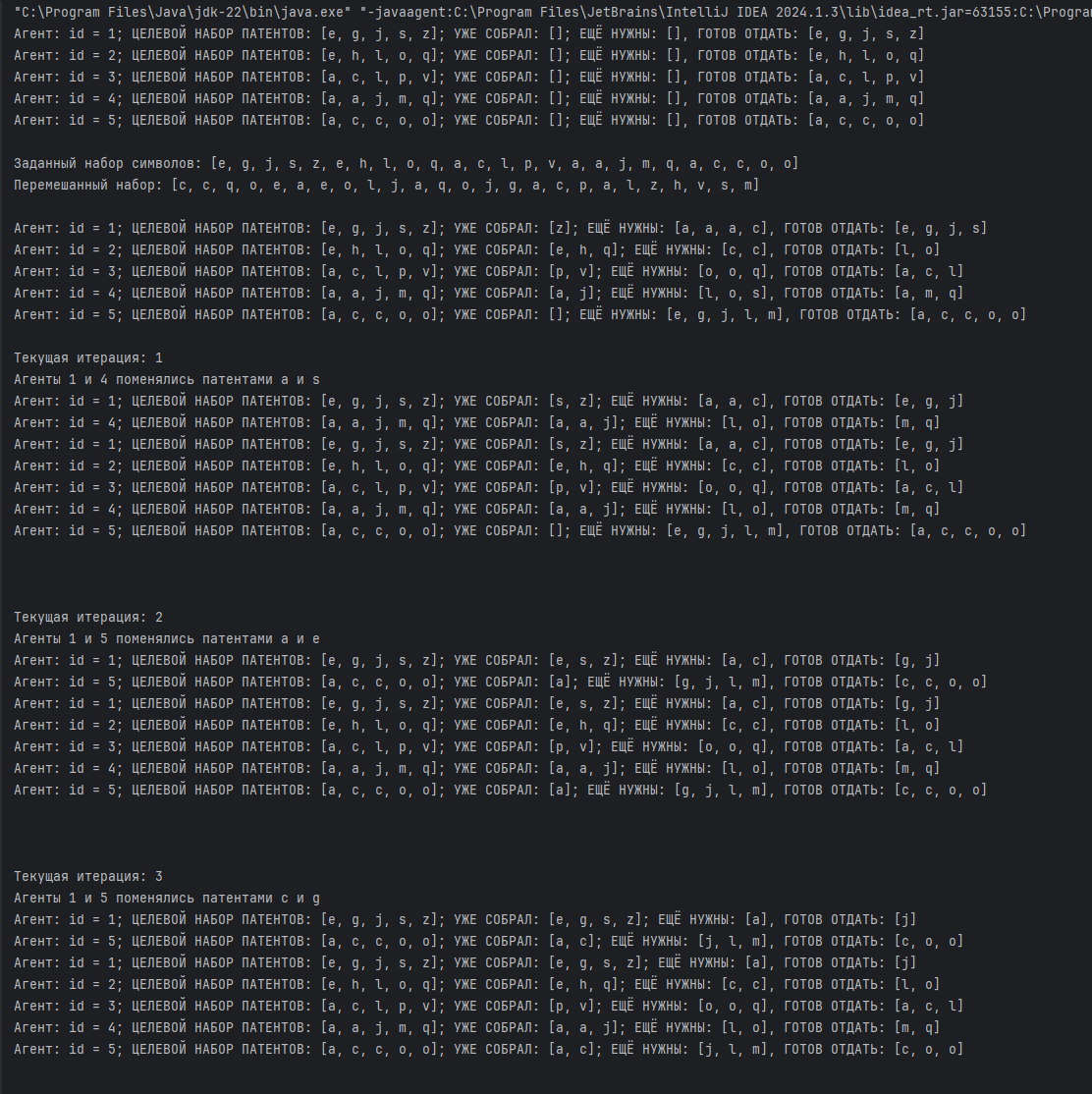
MessageBus.java

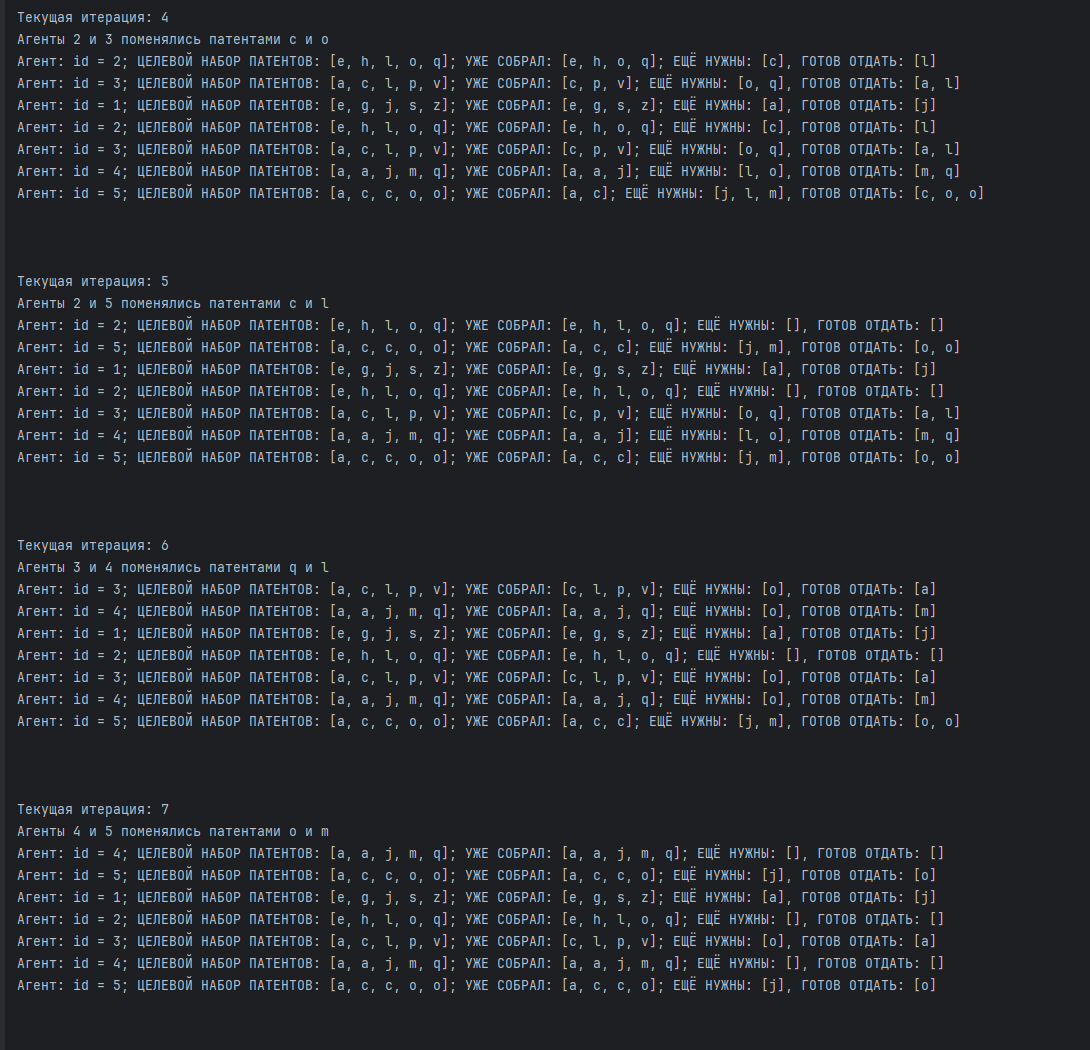
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class MessageBus {  
  
 public MessageBus() {}  
  
 public void consilium(List<Agent> listOfAgents) {  
 List<Map<String, Long>> listOfAllUselessPatents = new ArrayList<>();  
 boolean successfulTrade = false;  
  
 for (Agent agent : listOfAgents) {  
 Map<String, Long> patentsCount = agent.getUselessPatents().stream()  
 .collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*()));  
 listOfAllUselessPatents.add(patentsCount);  
 }  
  
 searchLoop:  
 for (int i = 0; i < listOfAgents.size(); i++) {  
 Map<String, Long> missingPatentsI = listOfAgents.get(i).checkingTask();  
 for (int j = 0; j < listOfAgents.size(); j++) {  
 if (i != j) {  
 if (missingPatentsI.keySet().stream().anyMatch(listOfAllUselessPatents.get(j)::containsKey)) {  
 Map<String, Long> missingPatentsJ = listOfAgents.get(j).checkingTask();  
 if (missingPatentsJ.keySet().stream().anyMatch(listOfAllUselessPatents.get(i)::containsKey)) {  
 trade(listOfAgents.get(i), listOfAgents.get(j));  
 listOfAllUselessPatents.set(i, listOfAgents.get(i).getUselessPatents().stream()  
 .collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*())));  
 listOfAllUselessPatents.set(j, listOfAgents.get(j).getUselessPatents().stream()  
 .collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*())));  
 successfulTrade = true;  
 break searchLoop;  
 }  
 }  
 }  
 }  
 }  
  
 if (!successfulTrade) {  
 for (Agent agent : listOfAgents) {  
 if (!agent.checkCompletion()) {  
 System.*out*.println("Для агента " + agent.getId() + " прямой обмен невозможен, ищем цепочку обменов");  
 List<Integer> chain = findChain(listOfAgents, agent, listOfAllUselessPatents);  
 if (chain != null) {  
 System.*out*.println("Цепочка нашлась: " + chain);  
 }  
 }  
 }  
 }  
 }  
  
 private void trade(Agent agent1, Agent agent2) {  
 String patent1 = performSwap(agent1, agent2);  
 String patent2 = performSwap(agent2, agent1);  
  
 System.*out*.println("Агенты " + agent1.getId() + " и " + agent2.getId() + " поменялись патентами " + patent1 + " и " + patent2);  
 agent1.getInfo();  
 agent2.getInfo();  
 }  
  
 private String performSwap(Agent fromAgent, Agent toAgent) {  
 String tradedPatent = null;  
 Iterator<String> iterator = fromAgent.getUselessPatents().iterator();  
  
 while (iterator.hasNext()) {  
 String patent = iterator.next();  
 if (toAgent.checkingTask().containsKey(patent)) {  
 iterator.remove();  
 toAgent.swap(patent);  
 tradedPatent = patent;  
 break;  
 }  
 }  
 return tradedPatent;  
 }  
  
 private List<Integer> findChain(List<Agent> listOfAgents, Agent startAgent, List<Map<String, Long>> listOfAllUselessPatents) {  
 System.*out*.println("Начинается поиск цепи для агента " + startAgent.getId());  
 Agent currentAgent = startAgent;  
 List<Integer> path = new ArrayList<>(Collections.*singletonList*(startAgent.getId()));  
 Set<Agent> visitedAgents = new HashSet<>();  
 Agent newCurrentAgent = findNextAgent(listOfAgents, currentAgent, path, visitedAgents);  
  
 while (!Objects.*equals*(path.get(0), path.get(path.size() - 1))) {  
 if (newCurrentAgent != null) {  
 newCurrentAgent = findNextAgent(listOfAgents, newCurrentAgent, path, visitedAgents);  
 } else {  
 System.*out*.println("Цепочка для агента " + startAgent + " не сформирована");  
 return null;  
 }  
 }  
  
 System.*out*.println("Нашлась цепочка агентов: " + path);  
 chainExchange(listOfAgents, path, listOfAllUselessPatents);  
 return path;  
 }  
  
 private Agent findNextAgent(List<Agent> listOfAgents, Agent currentAgent, List<Integer> path, Set<Agent> visitedAgents) {  
 for (String uselessPatent : currentAgent.getUselessPatents()) {  
 for (Agent agent : listOfAgents) {  
 if (!agent.equals(currentAgent) && !visitedAgents.contains(agent)) {  
 if (agent.checkingTask().containsKey(uselessPatent)) {  
 path.add(agent.getId());  
 visitedAgents.add(agent);  
 System.*out*.println("Следующий агент для цепочки найден: " + agent.getId());  
 return agent;  
 }  
 }  
 }  
 }  
 System.*out*.println("Подходящий агент для цепочки не найден");  
 return null;  
 }  
  
 private void chainExchange(List<Agent> listOfAgents, List<Integer> path, List<Map<String, Long>> listOfAllUselessPatents) {  
 for (int i = 0; i < path.size() - 1; i++) {  
 for (Agent agent1 : listOfAgents) {  
 if (agent1.getId() == path.get(i)) {  
 String uselessPatent = agent1.getUselessPatents().get(0); // костыль  
 agent1.getUselessPatents().remove(uselessPatent);  
 for (Agent agent2 : listOfAgents) {  
 if (agent2.getId() == path.get(i + 1)) {  
 agent2.swap(uselessPatent);  
 System.*out*.println("Агент " + agent1.getId() + " отдал патент " + uselessPatent + " агенту " + agent2.getId());  
 agent1.getInfo();  
 agent2.getInfo();  
 listOfAllUselessPatents.set(i, agent1.getUselessPatents().stream()  
 .collect(Collectors.*groupingBy*(e -> e, Collectors.*counting*())));  
 }  
 }  
 }  
 }  
 }  
 }  
  
}

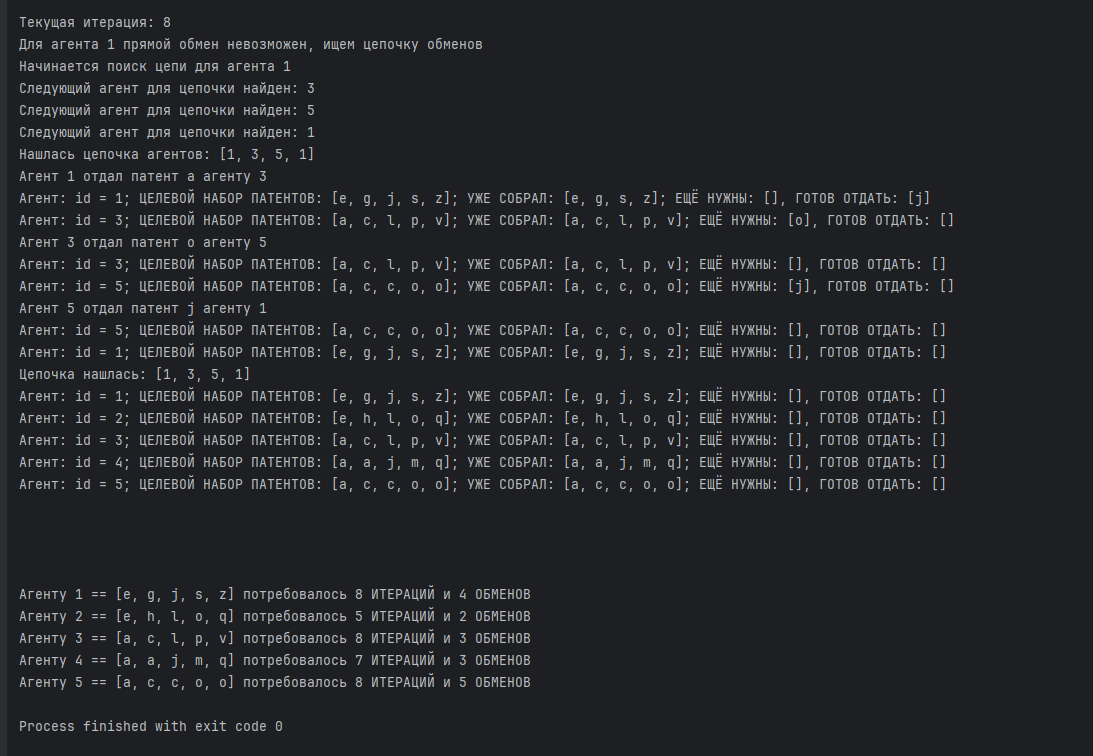
Main.java

import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
import java.util.Random;  
  
public class Main {  
  
 private static final int *NUMBER\_OF\_AGENTS* = 5;  
  
 private static int *timeOfStart* = 0;  
  
 private static List<Agent> *listOfAgents* = new ArrayList<>();  
  
 public static void main(String[] args) {  
 *createAgent*(*NUMBER\_OF\_AGENTS*, *listOfAgents*);  
 *startEnvironment*(*listOfAgents*, *timeOfStart*);  
 }  
  
 private static void createAgent(int NUMBER\_OF\_AGENTS, List<Agent> listOfAgents){  
 for (int i = 1; i <= NUMBER\_OF\_AGENTS; i++) {  
 Agent agent = new Agent(i);  
 agent.getInfo();  
 listOfAgents.add(agent);  
 }  
  
 List<String> allPatents = new ArrayList<>();  
 for (Agent agent : listOfAgents) {  
 allPatents.addAll(agent.getTargetTask());  
 }  
  
 System.*out*.println("\nЗаданный набор символов: " + allPatents);  
  
 Collections.*shuffle*(allPatents, new Random());  
 System.*out*.println("Перемешанный набор: " + allPatents + "\n");  
  
  
 int patentsPerAgent = allPatents.size() / NUMBER\_OF\_AGENTS;  
 for (int i = 0; i < patentsPerAgent; i++) {  
 for (Agent agent : listOfAgents) {  
 if (!allPatents.isEmpty()) {  
 String patent = allPatents.remove(0);  
 agent.obtainingAPatent(patent);  
 }  
 }  
 }  
  
  
 for (Agent agent : listOfAgents) {  
 agent.getInfo();  
 }  
 }  
  
 private static void startEnvironment(List<Agent> listOfAgents, int timeOfStart){  
 MessageBus messageBus = new MessageBus();  
  
 while (listOfAgents.stream().filter(agent -> agent.getTargetTask().size() == agent.getImportantPatents().size()).count() != listOfAgents.size()) {  
  
 timeOfStart++;  
 System.*out*.println("\nТекущая итерация: " + timeOfStart);  
 messageBus.consilium(listOfAgents);  
  
 // Перебор агентов  
 for (Agent agent : listOfAgents) {  
 if (agent.getTargetTask().size() != agent.getImportantPatents().size() || agent.getNumberOfIteration() == 0) {  
 agent.updateState();  
 }  
 agent.getInfo();  
 }  
  
 System.*out*.println("\n");  
  
 try {  
 Thread.*sleep*(500); // Удобство вывода  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 System.*out*.println("\n");  
  
 for (Agent agent : listOfAgents) {  
 agent.updateState();  
 agent.winInfo();  
 }  
  
 }  
  
}

**Примеры вывода**

****

****

****